VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2013 **David Vašíček**

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Vytvoření animací pro výpočet metody smyčkových proudů a řezových napětí

Creating Animations for the Calculation Method of Loop Currents and Cutting Voltages

2013 **David Vašíček**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **David Vašíček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Vytvoření animací pro výpočet metody smyčkových proudů a řezových napětí
Creating Animations for the Calculation Method of Loop Currents and Cutting Voltages

Zásady pro vypracování:

1. Seznámení se s problematikou metody smyčkových proudů a metody řezových napětí
2. Vytvoření interaktivní animace v programu Adobe Flash
   - základní výpočty
   - volba obvodu, volba stromu obvodu
   - výpočet metodou smyčkových proudů
   - výpočet metodou řezových napětí
   - zobrazení výsledků v podobě I. a II. Kirchhoffova zákona
   - ověření výsledků Tellegenovou větou

Seznam doporučené odborné literatury:

[1] M. Mikulec, V. Havlíček: Základy teorie elektrických obvodů, Praha 1997, ISBN 80-01-02519-5
[2] M. Mikulec: Teorie obvodů - přednášky, Praha 1989, ISBN 80-01-00147-4
[3] J. Székely: Teoretická elektrotechnika I, Žilina 1968, Skripta

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Stanislav Zajaczek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

prof. RNDr. Vladimír Vašinek, CSc.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: *07.05.2013*

……..……
podpis studenta

Poděkování

Rád bych poděkoval Ing. Stanislavu Zajaczkovi Ph.D. za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Bakalářská práce se zabývá vytvořením animace pro metodu smyčkových proudů a metody řezových napětí. Hlavním cílem této práce bylo vytvoření výpočetního programu, který by na základě rozboru zadaného elektrického obvodu, byl schopen stanovit jednotlivé větvové veličiny v obvodu, podle metody smyčkových proudů nebo metody řezových napětí. Tento projekt je realizován v programu Adobe Flash Professional a objektově orientovaném programovacím jazyce ActionScript3.

Klíčová slova

Animace, Adobe Flash Professional, ActionScript3, metoda smyčkových proudů, metoda řezových napětí

Abstract

This Bachelor thesis looks into a creation process of a loop currents and a cutting voltage methods animation. The main goal of the thesis was a creation of a computational program, which would be able to set individual branch values in a circuit by the method of loop currents or the cutting voltage method using analysis of the specified electrical circuit. This project is implemented in Adobe Flash Professional software and an object oriented programming language called ActionScript 3.

Key words

Animation, Adobe Flash Professional, ActionScript3, Method of Loop Current, Method Cutting Voltages

Seznam použitých symbolů

Symbol	Jednotky	Význam symbolu
U	V	Napětí
I	A	Proud
R	Ω	Rezistor
G	S	Elektrická vodivost

Seznam použitých zkratek

Zkratka	Význam
Swf	Shockwave Flash
fla	Soubor typu Flash
PDF	Portable Document Format
AS	AcrionScript
XML	Extensible Markup Language
OS	Operating System
HTML	HyperText Markup Language
R	Rezistor
IU	Ideální napěťový zdroj
II	Ideální proudový zdroj
SU	Skutečný napěťový zdroj
SI	Skutečný proudový zdroj
SUR	Vnitřní odpor skutečného napěťového zdroje
SIR	Vnitřní odpor skutečného proudového zdroje
MSP	Metoda smyčkových proudů
MŘN	Metoda řezových napětí

Obsah

1 Úvod

Tato Bakalářská práce se zabývá obecnými metodami řešení lineárních elektrických obvodů, konkrétně se jedná o metodu smyčkových proudů a metody řezových napětí. Tyto dvě metody jsou založeny na I. a II. Kirchoffově zákoně, díky nimž můžeme stanovit neznámé veličiny v elektrickém obvodu. Těmito neznámými veličinami se myslí dopočítání jednotlivých úbytků proudů, či napětí, na jednotlivých prvcích v obvodu.

Proto bylo mým hlavním úkolem vytvoření interaktivní animace, která by dokázala tyto úbytky na jednotlivých větvích vypočíst. Bylo velmi důležité tento projekt udělat tak, aby si sám uživatel mohl vybrat obvod, se kterým by chtěl pracovat. Poté dle tohoto zadaného obvodu sestavit jednotlivé rovnice pro jednotlivé smyčky, nebo řezy, dle I. a II. Kirchoffova zákona, sestavit matici právě podle těchto rovnic a vyjádřit úbytky proudů, či napětí na jednotlivých větvích obvodu.

Celá tato bakalářská práce je členěna do celkem sedmi kapitol. Po přeskočení této úvodní kapitoly přejdeme ke kapitole druhé, kde si objasníme, k čemu obecné metody pro řešení lineárních obvodů slouží, a jak tyto metody aplikovat v elektrickém obvodu a následně i v této práci. Následně si řekneme něco málo o programech, ve kterých budeme tuto práci realizovat a ukážeme si, jak tyto programy vypadají a jak pracují. Poté si ukážeme, jak byl cely projekt realizován, jakým způsobem byl řešen a jak jsem při této práci postupoval. V poslední řadě si ukážeme a vysvětlíme celou funkčnost výsledného programu, jak správně pracuje a jak správně v něm pracovat.

2 Teoretický rozbor

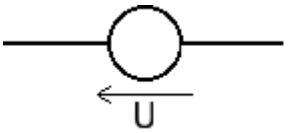
2.1 Obecné informace

Metody smyčkových proudů a řezových napětí slouží k obecnému řešení lineárních elektrických obvodů, které nám umožňují stanovit neznámé větvové veličiny elektrického obvodu. Tyto větvové veličiny můžeme vyjádřit buďto z proudu nezávislých smyček, nebo z napětí nezávislých řezů, jejichž volba vychází z topologického rozboru schématu obvodu. Každá z těchto metod je založena na analýze dle Kirchhoffových zákonů.

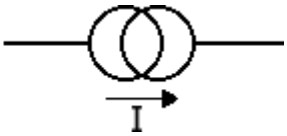
Pro snadnější řešení lze použít i metody postupného zjednodušování větví obvodu a transfiguraci větví s reálnými zdroji napětí na zdroje proudu a naopak. Výhodou těchto metod, oproti Kirchhoffových zákonů, je snížení počtu obvodových rovnic a celkové zjednodušení výpočtů. Pro výběr metody je důležité zohlednit počet proudových a napěťových větví, jelikož u metody smyčkových proudů počet větví snižuje počet proudů nezávislých větví a u metody řezových napětí počet neznámých napětí nezávislých řezů. Jeli rozdíl počtu nezávislých větví a počtu proudových větví obvodu menší než rozdíl počtu nezávislých uzlů a počtu napěťových větví obvodu, použijeme metodu smyčkových proudů. V opačném případě, kdy je rozdíl počtu nezávislých uzlů a počtu napěťových větví obvodu menší než rozdíl počtu nezávislých větví a počtu proudových větví obvodu použijeme metodu řezových napětí. [1]

Během studií těchto textů, se můžete setkat s následujícími pojmy:

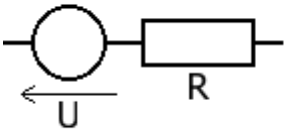
- *Uzel* - bod, ve kterém se stýkají alespoň dva vodiče
- *Počet nezávislých uzlů* – je o jeden uzel menší, než celkový počet uzlu v obvodu
- *Větev* - spoj mezi právě dvěma uzly, který je tvořený jedním prvkem, nebo více prvky spojenými za sebou
- *Počet větví stromu* – je roven počtu nezávislých uzlů
- *Počet nezávislých větví* – je roven celkový počet větví obvodu bez počtu větví stromu
- *Smyčka* - uzavřená část elektrického obvodu tvořena alespoň třemi větvemi
- *Ideální zdroj napětí* – Nezávisle na odebíraném proudu jeho napětí na svorkách neklesá. Obrázek 2.1. [5]
- *Ideální zdroj proudu* – Nezávisle na tom, jak velké napětí tento zdroj musí vyvinout, dodává do obvodu stále daný proud. Obrázek 2.2. [5]
- *Skutečný zdroj napětí* – Při odběru proudu jeho svorkové napětí klesá. Můžeme jej nahradit ideálním zdrojem stálého napěti, ke kterému bychom sériově připojili rezistor o odporu. Obrázek 2.3. [5]
- *Skutečný zdroj proudu* – Dle toho, jak velké napětí tento zdroj musí vyvinout, klesá proud dodávány do obvodu. Můžeme jej nahradit ideálním zdrojem stálého proudu, ke kterému bychom paralelně připojili rezistor o odporu. Obrázek 2.4. [5]



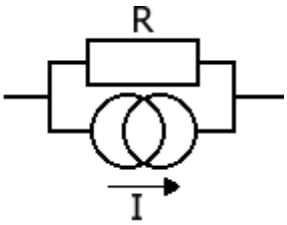
Obrázek 2.1: Ideální zdroj napětí



Obrázek 2.2: Ideální zdroj proudu



Obrázek 2.3: Skutečný zdroj napětí



Obrázek 2.4: Skutečný zdroj proudu

2.2 Metoda smyčkových proudů

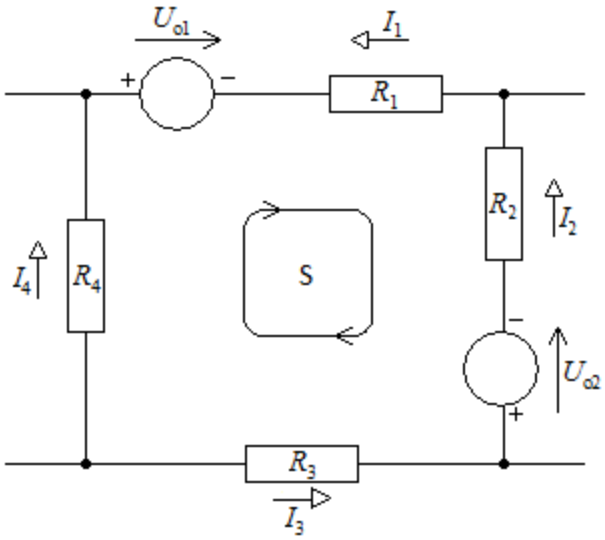
Tato metoda slouží k určení neznámých větvových proudů v obvodu a využívá 2. Kirchhoffův zákon. Tento zákon byl popsán německým fyzikem Gustavem Robertem Kirchhoffem v roce 1845. Mezi práce tohoto fyzika patří krom 2. Kirchhoffůva zákona i zákon první, kterým se ale budeme zabývat až v kapitole 2.3. [6]

2. Kirchhoffův zákon praví, že součet úbytků napětí na spotřebičích se v uzavřené části obvodu, ve smyčce, rovná součtu elektromotorických napětí zdrojů v této části obvodu. Jinak též řečeno, algebraický součet všech svorkových napětí zdrojů a všech úbytků napětí na spotřebičích se v uzavřené smyčce rovná nule. Toto vyjádření pak můžeme vidět v rovnici 2.1. [7]

$$U_1 + U_2 + ... + U_n = 0 \qquad \sum_{k=1}^{n} U_k = 0 \tag{2.1}$$

Pro bližší pochopení si ukážeme názorný příklad, který můžeme vidět na obrázku 2.5. Zde můžeme vidět část elektrického obvodu, respektive jednu uzavřenou smyčku, která se skládá ze 4 rezistorů a dvou zdrojů elektrického napětí. V prvé řadě řešení tohoto obvodu si musíme vyjádřit úbytky napětí, na jednotlivých spotřebičích, v našem případě U_{R1} až U_{R4}. Tyto jednotlivé úbytky

následně vyjádříme dle Ohmova zákona, tudíž úbytek napětí je roven součinu hodnoty odporu na příslušném rezistoru a proudu, který tímto rezistorem protéká. Matematicky tedy zapsáno $U = R \cdot I$.



Obrázek 2.5: Část elektrického obvodu s vyznačenou smyčkou

Jak už jsme si řekli o odstavec výše, v prvé řadě bylo potřeba vyjádřit si jednotlivé úbytky napětí ve smyčce, kterou nám tento obvod tvoří. Rovnici budeme sestavovat podle směru hodinových ručiček a to proto, že v obvodu, pod označením S, je tohle zaznačeno. Jinak bychom si samozřejmě mohli sami zvolit, jakým směrem by smyčka mohla v obvodu proházet. Následně podle tohoto směru budeme určovat, jestliže jednotlivé úbytky napětí budou kladné, nebo záporné. Tohle bude ovlivněno směrem napětí na každém spotřebiči, zdali šipka tohoto zdroje je orientována proti směru smyčky, nebo ve směru stejném. Pokud tedy začneme procházet celou tuto smyčku, začínajíce zdrojem elektrického napětí U1, dospějeme k rovnici 2.2.

$$U_1 - U_{R1} - U_{R2} - U_2 - U_{R3} + U_{R4} = 0 \tag{2.2}$$

Pokud následně nahradíme veškeré úbytky napětí dle Ohmova zákona, tak jak jsme si už jednou popisovali, dospějeme k závěru rovnice 2.3.

$$U_1 - R_1 \cdot I_1 - R_2 \cdot I_2 - U_2 - R_3 \cdot I_3 + R_4 \cdot I_4 = 0 \tag{2.3}$$

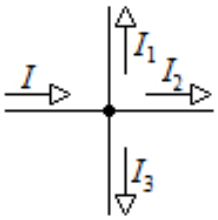
Pokud takto postupujeme v celém obvodu, a tyto jednotlivé rovnice pro jednotlivé smyčky vyjádříme, jsme schopni sestavit *n* rovnic o *n* neznámých, ze kterých následně můžeme vyjádřit neznámé proudy v těchto smyčkách.

2.3 Metoda řezových napětí

Cílem této metody je stanovení neznámých větvových napětí obvodu. Na rozdíl od metody smyčkových proudů nevychází z 2. Kirchhoffova zákonu, ale ze zobecněného zákonu prvního. Ten popisuje zákon o zachování elektrického náboje a praví, že algebraický součet všech proudů vtékajících do uzlů se rovná součtu proudů, které z uzlu vytékají. Jinak taky řečeno, algebraický součet všech proudu v uzlu je roven nule. Toto vyjádření pak můžeme vidět v rovnici 2.4. [7]

$$I_1 + I_2 + ... + I_{\mathrm{n}} = 0 \qquad \sum_{k=1}^{n} I_k = 0 \tag{2.4}$$

Pro bližší pochopení se můžeme podívat na obrázek 2.6, kde máme část elektrického obvodu, konkrétně jeden uzel se čtyřmi větvemi. Jak můžeme vidět, z jedné větve vstupuje do uzlu proud *I* a v ostatních větví následně z uzlu vystupují tři proudy I_1-I_3. Pro tyto proudy platí, že ty proudy, které do uzlu vstupují, jsou proudy kladné a ty co z uzlu vystupují, jsou proudy záporné. Proto tedy můžeme sestavit rovnici 2.5.

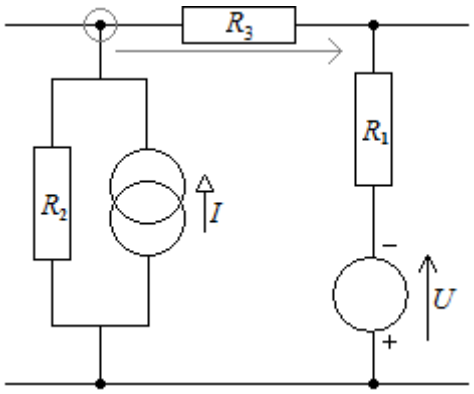


Obrázek 2.6: Grafické znázornění 1.Kirchoffova zákona

$$I - I_1 - I_2 - I_3 = 0 \;\Rightarrow\; I = I_1 + I_2 + I_3 \tag{2.5}$$

V případě řezových napětí dosazujeme za proudy pasivních větví definiční vztahy mezi větvovým proudem a napětím. Jednotlivá znaménka proudů pak orientujeme podle jejich počátečních šipek vůči zvolené referenční orientaci proudu řezu. Z pravidla proudu vytékajícímu z řezu odpovídá znaménko plus a proudu vytékajícímu znaménko mínus. [1]

Na obrázku 2.7 se můžeme podívat na část elektrického obvodu, kde v levém horním uzlu máme zaznačené šedé kolečko, což značí řez a šedá šipka vycházející z tohoto řezu značí napětí, které z řezu vytéká. Tento řez jsme volili náhodně a uživatel si může sám zvolit jednotlivé řezy, podle kterých bude obvod řešen. Pokud bychom tedy začali sestavovat rovnici pro tento řez, budeme sledovat jednotlivé větve, které uzlu v daném řezu přísluší. Protože na jedné větvi nevíme, co se na této větvi nalézá, tuto větev necháme prázdnou.



Obrázek 2.7: Část elektrického obvodu s vyznačeným řezem

V prvé řadě se podíváme na větev, ve které se vyskytuje skutečný proudový zdroj. Podle definice 1.Kirchoffova zákona, by bylo zřejmé, že tento proud na tomto zdroji bude kladný, jelikož do uzlu vtéká. To však není úplně pravdivé. V této metodě musíme sledovat jednotlivé proudy, které z jednotlivých řezů vytékají, a porovnávat je s proudy na jednotlivých prvcích. Proto tento proud bude záporný, jelikož teče proti proudu, který vytéká z řezu. V rovnici 2.6 se můžeme podívat, jak tedy tato rovnice pro tento řez bude vypadat.

$$I_{G2} - I_2 + I_{G3} = 0 \tag{2.6}$$

Pokud následně tuto rovnici upravíme a jednotlivé úbytky proudů na rezistorech převedeme dle Ohmova zákona ve tvaru $I = G \cdot U$, kde G značí elektrickou vodivost, která je rovna obrácené hodnotě rezistoru. Rovnice bude tedy vypadat dle rovnice 2.7.

$$\frac{1}{R_2} - I_2 + \frac{1}{R_3} = 0 \tag{2.7}$$

Tak jako u metody smyčkových proudů, tak i u této metody, takto postupujeme v celém obvodu a pro jednotlivé řezy vyjádříme jednotlivé rovnice. Ze kterých jsme následně schopni sestavit *n* rovnic o *n* neznámých, ze kterých dokážeme vyjádřit neznáme napětí v jednotlivých řezech.

3 Vývojové prostředí

3.1 Adobe Flash Professional

Adobe Flash Prefessional je softwarova platforma pro tvorbu grafických vektorových programů, která byla dříve ve vlastnictví Macromedia, avšak v dnešní době patří společnosti Adobe Systém. Ta je softwarovou firmou zabývající se především vývojem počítačové grafiky a je známá jako autor standardů PostScript nebo také PDF. Mezi nejznámější práce těchto autorů patří především Adobe Photoshop, Adobe Illustrator, Adobe Acrobat nebo Adobe Reader, který slouží právě ke čtení, dnes snad nejznámějším, grafických dokumentům PDF.

Adobe Flash lze v dnešní době uplatnit především u vytváření vektorových grafik, tvorbě animací, her, nebo také internetových aplikací, které můžeme spouštět pomocí programu Adobe Flash Player. Tyto aplikace a animace pak můžeme programovat za pomocí objektově-orientovaného jazyka, dříve již zmiňovaného, ActionScript. Jeho výhodou je výsledná velikost souborů, která je velmi malá a proto dnes již zcela nahradila dříve používané formáty GIF. [8] [9]

3.2 ActionScript3

Action Script, neboli také velmi často zkráceně AS je programovací jazyk pro aplikace, které jsou vyvíjeny především pomocí Adobe Flash, ale také i jinými vývojářskými nástroji, které využívají stejných datových formátů. Tento jazyk vychází ze standardizované verze programovacího jazyka JavaScript nazývaný ECMAScript.

Action Script byl prvotně navržen, jako skriptovací jazyk v té době ještě ve vlastnictví Adobe Macromedia a byl především určen pro ovládání jednoduchých 2D vektorových animací. Tento jazyk byl však velice omezený ve všech směrech. Proto do něj vývojáři vložili jednoduchý příkaz „action“, ze Scriptu se stal Action Script a ten začal např. umožňovat připojit k tlačítku jednotlivé události na kliknutí myši a tak podobně.

Nyní je nejnovější verzí tohoto programovacího jazyka verze 3. Oproti předchozím verzím se tato verze kompiluje mnohokrát rychleji, nabízí lepší zpracování XML, ale požaduje po uživatelích lepší znalosti objektově orientovaného programování. ActioScript3 vývojáři přidali do programu Flash Player až ve verzi 9, tudíž je tento jazyk podporovaný v přehrávačích vyšších verzí nebo právě ve verzi 9. [10]

4 Postup práce

4.1 Vytvoření nového projektu

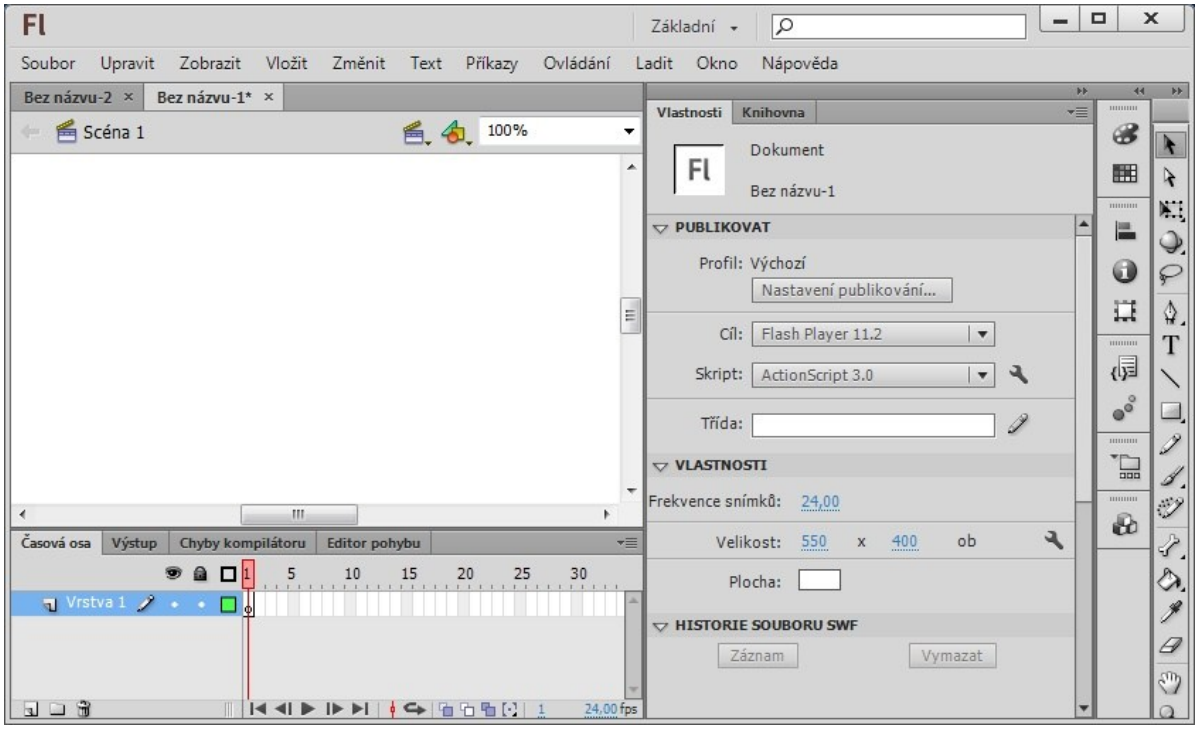
Při spuštění programu Adobe Flash Professional nám program nabízí vytvoření nového souboru Flash (*.fla) s možnosti nastavením publikování např. pro ActionScript2, AsctionSript3, prostředí AIR, jak pro Android, tak pro iOS, nebo nám umožňuje vytvoření nových tříd, které pak můžeme v projektu využít. V tomto okně také můžeme pro nový projekt rovnou nastavit rozměry plátna, jednotky pravítka, barvu pozadí, nebo také nastavení automatického ukládání projektu vždy po deseti minutách.

Pokud by uživatel nový projekt vytvářet nechtěl, program Adobe Flash nabízí již několik připravených předloh, které si uživatel může otevřít a poté jen upravovat, dle vlastních potřeb. V mém případě jsem vytvořil úplně nový projekt s nastaveným publikováním pro ActionScript3. Nyní se mi otevřelo zcela nové okno, ve kterém jsem mohl začít realizovat svůj projekt.

4.2 Seznámení se s programem Adobe Flash Professional

Prvním krokem při zahájení mé práce bylo zapotřebí, seznámit se s vývojovým prostředím programu Adobe Flash Professional s kterým jsem se v minulosti už párkrát setkal, pouze však jen z ukázky mých přátel, kteří v tomto programu již pracovali. Tento program se napovrch jeví jako celkem jednoduchý a příjemný program a dokonce i po hlubším zkoumání tento názor přetrvává. Proto nebyl žádný problém se s tímto programem velmi rychle naučit pracovat, a velmi dobře se v něm začít orientovat.

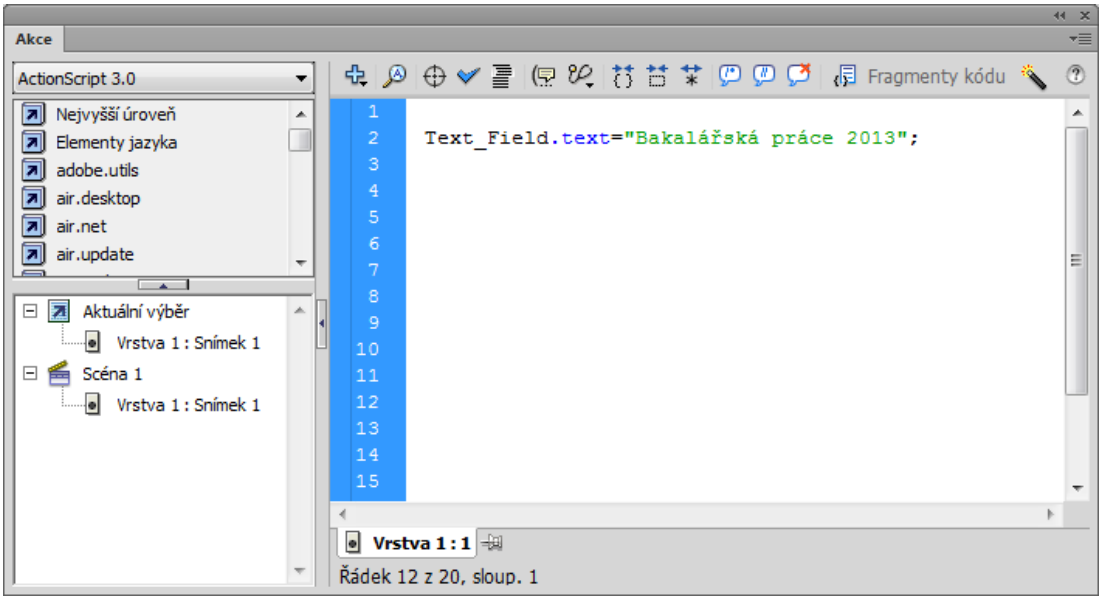
Pokud se podíváme na obrázek 4.1, můžeme vidět prostředí právě tohoto programu. Největší prostor na obrazovce nám tvoří kreslící plátno umístěné v levé části pracovní plochy a neliší se o žádné velké rozdíly oproti Windowsovskému programu Malování. Abychom však mohli v tomto plátně pracovat, budeme potřebovat kreslící nástroje, které můžeme najít v panelu napravo obrazovky, který se skládá se ze všech důležitých nástrojů, které bychom mohli využívat. Mezi tyto nástroje patří úsečka, elipsa, čtverec, tužka, text a mnoho dalších. Na obrazovce se také nachází okno vlastnosti, kde poté můžeme definovat například přesné rozměry těchto nakreslených objektů, nebo u textových polí styl písma, barvu písma nebo velikost písma. V poslední řadě můžeme ve spodní části obrazovky nalézt panel, který nám nabízí Časovou osu, Chyby kompilátoru nebo Výstup. Pro program Adobe Flash Professional je tato časová osa velmi důležitá z toho důvodů, že na ni může uživatel definovat, v kterou chvíli se budou skrývat nebo zobrazovat jednotlivé objekty na plátně, samotný průběh animace apod.



Obrázek 4.1: Prostředí programu Adobe Flash Prtofessional

Tento program již v sobě nabízí objektově orientovaný programovací jazyk ActionScript, tím se tedy nemusí instalovat žádné jiné programy a je velmi rychle přístupný z hlavní obrazovky programu. K takovému spuštění tohoto editoru slouží tlačítko *F9,* nebo se k němu také můžeme dostat pomocí záložky *Okno*, kde se tento editor skrývá pod názvem *Akce*.

Tento editor můžeme vidět na obrázku 4.2. Editor se nijak neliší od klasických editorů pro psaní kódu. Největší místo na této obrazovce tvoří část, ve které se píše samotný kód programu, kde v modrém sloupci máme označeny jednotlivé řádky editoru, a úplně naspod je zobrazeno, na kterém řádku se uživatel nachází, a kolik řádku kódu se v editoru nalézá. Toto mnohonásobně ulehčuje práci uživateli, pokud by udělal v kódu chybu, program mu zobrazí číslo řádku, na kterém se chyba nalézá. Proto uživatel nemusí zdlouhavě hledat svou chybu, pokud píše program o hodně řádcích. Editor také nabízí i další funkce, které se nalézají v horní části tohoto editoru. Ty slouží např. k hledání, vložení fragmentu kódu, formátování kódu, sbaleni kódu do závorek apod. Jedná se tedy o funkce usnadňující uživateli svou práci při vyvíjení programů.

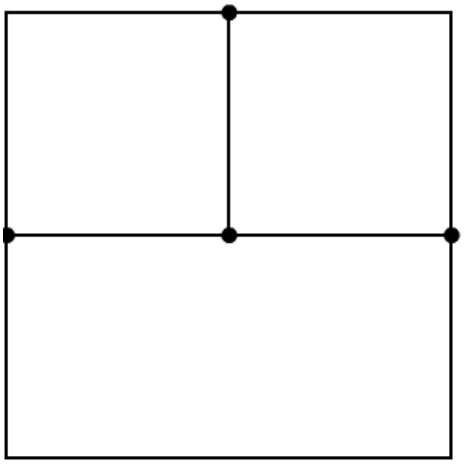


Obrázek 4.2: Prostředí editoru pro psaní kódu

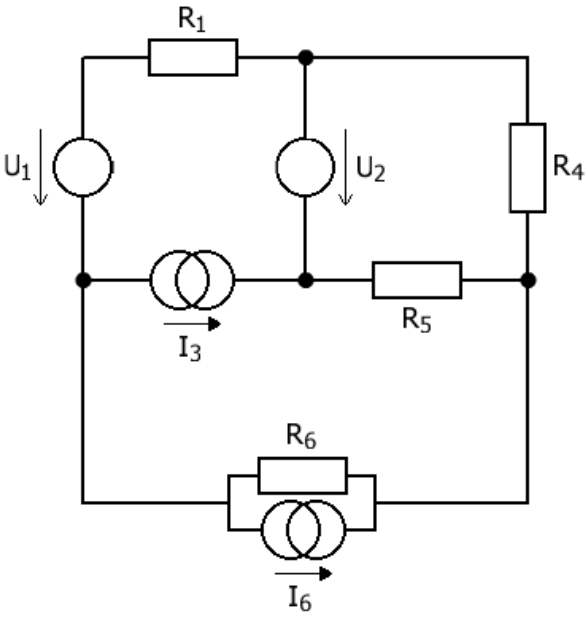
4.3 Tvorba kostry obvodu

Po seznámení se s programem, a prostudování veškeré potřebné látky, bylo ještě potřeba domluvit se na určitých požadavcích, které by program měl splňovat. Tím zásadním bylo umožnění uživatelům, sami si zvolit elektrický obvod, se kterým by chtěli pracovat. Následně po předběžném návrhu pár konceptů na papír, jsem mohl začít celý projekt realizovat.

Nejprve bylo zapotřebí vytvořit základní kostru obvodu, která se v mém případě skládala z šesti větví a čtyř uzlů. Tuto kostru jsem realizoval tak, že jsem složil několik vertikálních a horizontálních úseček, jejichž výsledný obrazec utvořil celou kostru obvodu a jednotlivé úsečky pak tvořily samotné větve, což můžeme vidět na obrázku 4.3. Tyto úsečky jsem do projektu importoval jako obrázky, z důvodu jednoduššího přístupu k těmto objektům uvnitř programu. Jakmile byla kostra obvodu zhotovena, bylo zapotřebí provést, aby se každá větev v tomto obvodu stala aktivní. To bylo možné realizovat tak, že bych do obvodu přidal nějaká funkční tlačítka, která by byla schopna provádět určité akce. Proto jsem do projektu importoval několik průhledných obrázků, které jsem následně převedl na tlačítka a umístil je na pozice jednotlivých větví. Tímto jsem skryl tlačítka a přitom zůstaly zobrazeny jednotlivé větve. Teď už stačilo jednotlivým tlačítkům přiřadit vše to, co by měly provádět a začít psát samotný program v ActionScript.



Obrázek 4.3: Základní kostra obvodu



Obrázek 4.4: Správně zadaný obvod

Abych mohl každému tlačítku definovat to, co přesně má provádět, musel jsem každému takovému tlačítku nastavit tzv. posluchače a metodu *MouseEvent.CLICK*, kde každý posluchač sleduje případ, kdy se na tlačítko klikne myší. Takový kód pak můžeme vidět na obrázku 4.5, na kterém písmeno *A* značí název tlačítka, *addEventListener* vytvořilo na toto tlačítko posluchače, který kontroluje kliknutí myší, což definuje příkaz v první části závorky. V druhé části je pak název funkce, která se má provést, pokud na dané tlačítko myší kliknu. V tady tomto případě se jedna o funkci *VetevA*, která je popsána na druhém řádku obrázku 4.5 v příkazu *function*, kde si můžeme, všimnou tří teček, které se nalézají uprostřed složených závorek a na místo kterých přijde definice toho, co tlačítko má provádět.

```
A.addEventListener(MouseEvent.CLICK, VetevA);
function VetevA(event:MouseEvent):void{...}
```

Obrázek 4.5: Fragment kódu s nastavením posluchače pro větev A

Taková funkce by měla umožnit uživateli zobrazit jednotlivé prvky, které pak může do obvodu vkládat. Proto jsem do projektu importoval nové obrázky, tentokráte s vyobrazením jednotlivých prvků a to tedy s rezistorem, ideálním proudovým zdrojem, ideálním napěťovým zdrojem, skutečným proudovým zdrojem a skutečným napěťovým zdrojem. Tyto obrázky bylo nutné také převést na jednotlivá tlačítka, přiřadit jim posluchače a nadefinovat, co mají provádět. Kliknutím

na jakýkoli prvek se následně přidal daný prvek do obvodu, proto bylo zapotřebí importovat ještě do projektu další obrázky, které by vyobrazovaly jednotlivé prvky v každé větvi obvodu.

Na obrázku 4.6 můžeme vidět, právě přiřazení těchto posluchačů na jednotlivé prvky, které jsem v tomto případě prováděl jinak, než tomu bylo u nastavení posluchačů pro tlačítka jednotlivých větví. V prvních pěti řádcích máme přiřazení posluchačů jednotlivým tlačítkům s metodou *MouseEvent.CLICK* a s názvem funkce *Click*, která je přiřazena všem těmto tlačítkům. Výhodou tohoto řešení, je mnohonásobně zjednodušený kód programu, kde nemusíme vytvářet pět různých funkci, ale stačí nám pouze jedna. Abychom jsme však zjistili, na jaké to tlačítko jsme vůbec klikli, použil jsem takzvaný přepínač, *Switch*, který se bude přepínat, podle jména tlačítka, na které se klikne. To nám určuje příkaz *event.currentTarget.name*. Teď už nezbývá nic jiného než vytvořit jednotlivé případy *Case* pro tyto tlačítka, které budou provádět potřebné příkazy. Tyto příkazy přijdou na místo tří teček, které jsou ukončeny příkazem *break*. Na tomto obrázku můžeme vidět pouze dva tyto příkazy Case, ale zbylé tři se liší pouze v názvu jednotlivých tlačítek. V závěru to vypadá tak, že pokud zmáčknu nějaké tlačítko, zjistí se jeho jméno a právě podle tohoto jména se následně provede patřičný *Case*.

```
Rezistor.addEventListener(MouseEvent.CLICK, Click);
Idealni_zdroj_napeti.addEventListener(MouseEvent.CLICK, Click);
Idealni_zdroj_proudu.addEventListener(MouseEvent.CLICK, Click);
Skutecny_zdroj_napeti.addEventListener(MouseEvent.CLICK, Click);
Skutecny_zdroj_proudu.addEventListener(MouseEvent.CLICK, Click);

function Click(event:MouseEvent):void{
    switch (event.currentTarget.name){

        case "Rezistor" :
            ...
            break;

        case "Idealni_zdroj_napeti" :
            ...
            break;
```

Obrázek 4.6: Fragment kódu s nastavením posluchače pro jednotlivé prvky

Při realizaci této části vznikl první složitý problém, kdy se nějakému tlačítku nastavila metoda *Click* a následně se na toto tlačítko kliklo, tlačítko zůstalo napořád aktivní. Když se tedy následně kliklo na tlačítko jiné, označilo se i to tlačítko na, které bylo kliknuto již dříve. Tím se provedla akce pro všechny tlačítka stejná podle funkce, přiřazené naposledy zmáčknutému tlačítku. Problém byl v tom, že po ukončení této metody *Click*, na kterou jsem přidával posluchače, bylo potřeba posluchače taky odebrat. Realizaci můžeme vidět na obrázku 4.7.

```
Rezistor.removeEventListener(MouseEvent.CLICK, Click);
Idealni_zdroj_napeti.removeEventListener(MouseEvent.CLICK, Click);
Idealni_zdroj_proudu.removeEventListener(MouseEvent.CLICK, Click);
Skutecny_zdroj_napeti.removeEventListener(MouseEvent.CLICK, Click);
Skutecny_zdroj_proudu.removeEventListener(MouseEvent.CLICK, Click);
```

Obrázek 4.7: Fragment kódu s odebráním posluchače pro jednotlivé prvky

Další problém, který jsem zde řešil, byl vtom, že pokud jsem jednotlivé větve, tlačítka, a obrázky prvku vložil hromadně na jedno písmo, byl kurzor myši na tlačítku neaktivní, jelikož jej ostatní prvky překreslovaly a tedy na tlačítko nešlo kliknout. Proto bylo možné tyto tlačítka umístit napovrch všech vrstev, ale při skrývání a zobrazování jednotlivých prvků se mohlo stát, že by se napovrch dostal jiný prvek, než potřebné tlačítko a tlačítko se stalo opět neaktivní. Proto jsem použil příkaz *mouseEnabled*, jehož použití můžeme vidět na obrázku 4.8 a tento příkaz nám vypne kurzor myši na daný prvek. Chová se, jako by prvek na místě vůbec nebyl.

```
Prvek.mouseEnabled = false;
```

Obrázek 4.8: Ukázka kódu pro odebrání kurzoru myši na určitý prvek

Výsledek poté vypadá tak, že pokud uživatel spustí program, vyobrazí se základní kostra obvodu a uživatel si vybere v každé větvi, s kterým prvkem by chtěl počítat. Tudíž pokud označí nějakou větev, zobrazí se mu napravo od této kostry všech pět prvků a podle toho, který si vybere, přibude do dané větve obvodu. Takovýto výstup můžeme vidět na obrázku 4.4.

4.4 Algoritmus výpočtů

Nejdůležitější části této práce, bylo správně vypočíst zadaný elektrický obvod na základě zadaných hodnot jednotlivých prvků, které se v obvodu nachází. Tyto hodnoty následně vyplňuje sám uživatel poté, co správně zadal elektrický obvod a program uživateli nabídl celkem 8 textových polí s popisem jednotlivých prvků. Tyto textové pole jsou seřazeny podle jednotlivých větví, kdy první dva řádky o třech sloupcích tvoří všech šest větví 1-6 a třetí poslední řádek tvoří dva sloupce s větvemi, ve kterých se nachází skutečný zdroj napěťový, nebo skutečný zdroj proudový a v tomto řádku jsou zobrazeny vnitřní odpory právě těchto zdrojů.

Aby však program mohl správně pracovat, bylo zapotřebí zabránění špatně zadaných vstupních hodnot uživatelem a tím i kolizím při výpočtech. Takto špatně zadaná hodnota, se myslí znak jiný, než znak číselný. To z důvodu takového, že všechny vstupní hodnoty, které uživatel zadá,

se následně převádějí do datového typu *Number*, což je téměř totožné s datovým typem *Float*, neboli číselný formát s plovoucí desetinnou čárkou. Pokud by se tedy měl nějaký zadaný znak, který je typu *Char*, převádět na číselný formát, došlo by k chybě a také si umíme představit, co by asi udělalo dělení nějakého čísla textovým řetězcem např. „Ahoj“.

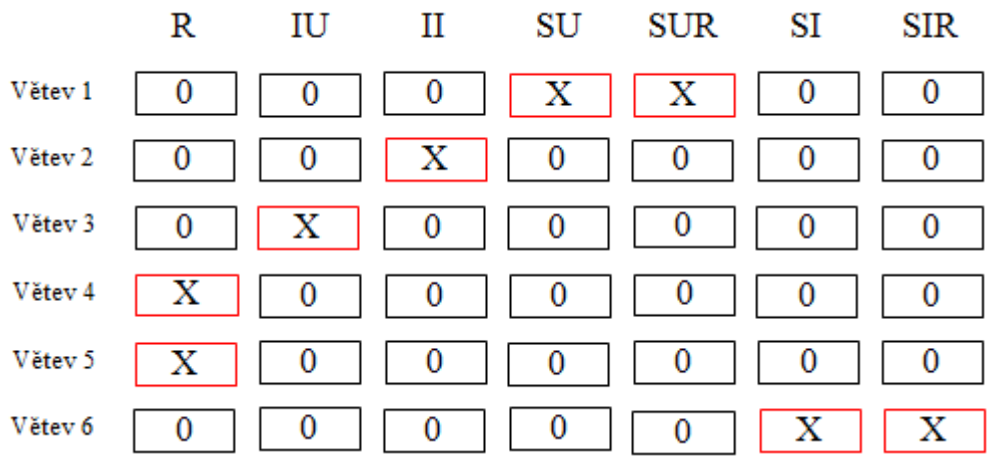
Z tohoto důvodu jsem uvnitř AS pro každé textové pole definoval příkaz *Restrict*, který můžeme vidět na obrázku 4.9. Tímto příkazem definujeme, co přesně do těchto polí můžeme vkládat za znaky. V této ukázce zdrojového kódu můžeme také vidět jméno daného textového pole, do kterého příkaz nastavit chceme, v našem případě se jedná o *Text_Field*. Následně mezi uvozovkami definujeme, které znaky jsou pro nás vhodné a které tedy povolíme. Pro nás stačily pouze znaky z numerické klávesnice, tudíž znaky *0* až *9* a znak *mínus*. V závěru to poté vypadá tak, že pokud povolíme jen nějaké znaky z tabulky ASCII, je klávesnice s kurzorem v daném textovém poli aktivní, pro ostatní znaky mimo tuto definovanou škálu je klávesnice neaktivní.

```
Text_Field.restrict = "-0-9";
```

Obrázek 4.9: Ukázka kódu pro nastavení znaků v textovém poli

Zde bylo velmi důležité právě tento znak *mínus* povolit, jelikož jak už jsem se jednou zmínil, veškeré počáteční směry toků proudů jsem defaultně volil nahoru a doprava a všechny počáteční toky napětí dolů a doleva. V případě, že nám tento směr nevyhovuje a uživatel by tedy lpěl na změně tohoto směru toku, použije právě tento znak *mínus* a vstupní číslo v textovém poli obrátí do záporné hodnoty. Tímto se směr toku na daném prvku obrátí. Krom všeho, je v obvodu povolena pouze desetinná tečka, pokud by uživatel chtěl zadat desetinné číslo. S desetinnou čárkou se v obvodu nepočítá.

Funkčnost všech výpočtů tohoto programu jsem založil na celkem 42 textových polí, kdy uživateli se zobrazí vždy pouhých 8 z nich, které už zde byly jednou zmíněny a to dle toho, které prvky se v obvodu nacházejí. Pokud by se tedy v obvodu např. nacházeli dva rezistory, jeden ideální napěťový zdroj, jeden ideální proudový zdroj, jeden skutečný napěťový zdroj a jeden skutečný proudový zdroj, v jednotlivých větvích, jak je tomu na obrázku 4.10, kde právě tyto prvky zobrazují jednotlivé červené rámečky, zobrazí se uživateli na obrazovce pouze tyto pole a uživatel do nich může vložit hodnotu. Ty pole, které do daného obvodu nepatří, zůstanou skryté a automaticky se do nich zapíše hodnota nula.



Obrázek 4.10: Uspořádání jednotlivých textových polí

Tento nápad mi následně velice pomohl při řešení jednotlivých kombinací výpočtů pro jednotlivé smyčky a řezy. Pokud se podrobněji podíváme na řešení metody smyčkových proudů, bylo zde potřeba dbát na umístění ideálního proudového zdroje v obvodu, podle kterého jsem následně sestavil tři základní kombinace smyček v obvodu, podle kterých byl celý obvod řešen. Jelikož se všechny prvky v jednotlivých smyčkách buď sčítají, nebo odčítají, mohl jsem vypočítat jakékoli kombinace výpočtů, aniž by mě zajímalo, co se na jednotlivých větvích nachází a k tomu jsem si dopomohl právě podle těchto 42 polí vyobrazených na obrázku 4.10. Pokud by tedy jedna ze smyček zahrnovala první tři větve obvodu, tak každá z těchto větví bude obsahovat všechny prvky, které se na těchto větvích mohou nacházet. Jedná se tedy o nějakých 125 kombinací zadání, jelikož jednotlivé vnitřní odpory jednotlivých skutečných zdrojů, se berou jako jeden prvek, i když je nutné je vypsat jako dvě textové pole. Následně taková logika může vypadat, tak jako je tomu na obrázku 4.11.

$$\text{Větev1} + \text{Větev2} + \text{Větev3} = 0$$
$$(R_1 - IU_1 + II_1 - SU_1 + SUR_1 + SI_1 + SIR_1) + (R_2 + IU_2 - II_2 + SU_2 + SUR_2 - SI_2 + SIR_2) +$$
$$+ (R_3 + IU_3 - II_3 + SU_3 + SUR_3 - SI_3 + SIR_3) = 0$$
$$(0 - 0 + 0 - X + X + 0 + 0) + (0 + 0 - X + 0 + 0 - 0 + 0) + (0 + X - 0 + 0 + 0 - 0 + 0) = 0$$

Obrázek 4.11: Funkčnost jednotlivých výpočtů

V prvním řádku této rovnice můžeme vidět tři větve, ze kterých se smyčka skládá, tudíž Větev 1, Větev 2, Větev 3. V druhém řádku jsou vypsány veškeré prvky, které se v dané větvi mohou, vyskytnou. Zde bylo potřeba uvědomit si, zdali se budou jednotlivé zdroje odčítat, nebo sčítat. Jelikož jsem ale veškeré toky i smyčky volil vždy stejným směrem, nebyl žádný problém rozpoznat znaménko, jaké jednotlivým zdrojům náleží. V dalším řádku už můžeme vidět přiřazení jednotlivých

hodnot jednotlivým prvkům dle obrázku 4.10. Tudíž se přiřadí pouze hodnoty těch prvků, které se na dané větvi nacházejí, a ostatním prvkům se přiřadí hodnota 0.

Takto jsem postupoval i u metody řezových napětí. Zde se však počítalo s elektrickou vodivostí, což je obrácená hodnota rezistoru. Bylo tedy nutné ošetřit případy, kdy by se rezistor ve větvi neobjevil, tím by se mu přiřadila hodnota 0 a při výpočtu elektrické vodivosti by nastalo dělení nulou. Proto jsem musel ošetřit případy, kdyby hodnota rezistoru nulová nebyla a buď ji správně vydělit, nebo až následně v závěru přiřadit výsledku vodivosti hodnotu 0. Oproti metodě smyčkových proudů jsem zde počítal s šesti základními kombinacemi obvodu, podle kterých jsem jednotlivé řezy řešil.

V této fázi jsem věděl, jaké hodnoty jednotlivé prvky mají a také, jaké prvky se na jednotlivých větvích nacházejí. Nezbývalo nic jiného, než z jednotlivých smyček nebo řezů, vyjádřit neznámé. Z tohoto důvodu jsem v projektu vytvořil objekt s názvem *DataGrid*, což je jakási tabulka, ve které jsem definoval potřebný počet řádků a sloupců. Následně jsem jednotlivé buňky tabulky naplnil hodnotami podle způsobů řešení jednotlivých smyček nebo řezů. Tato tabulka se mi následně proměnila v matici ve tvaru 3x3 * 1x3 = 1x3, ze které nebyl žádný problém neznámé vyjádřit.

Pro tuto část, jsem vytvořil algoritmus, který sleduje, v jaké smyčce se nachází ideální proudový zdroj, nebo ideální napěťový zdroj, jejíž hodnoty nám sám uživatel zadal. Jakmile zjistí, že se v některém z řádku tato definovaná hodnota nalézá, algoritmus řádek škrkne, k tomu i příslušný sloupec a z matice se stane matice ve tvaru 2x2 * 1x2 = 1x2, tedy dvě rovnice o dvou neznámých, pro které bylo důležité sestavit další algoritmus, který umí tyto rovnice vyřešit. Tato myšlenka vkládat jednotlivé hodnoty do tabulky mě napadla, při řešení příkladů na papír. Zdálo se být jednodušší, naplnit tabulku daty a poté přistupovat pouze k jejím buňkám a s nimi provádět jednotlivé operace, než přistupovat k velkému množství hodnot a pamatovat si, co přesně přičíst, odečíst, vydělit či vynásobit.

V této chvíli byly vyjádřeny jednotlivé neznáme v jednotlivých smyčkách, či řezech, ze kterých už nebyl žádný problém vyjádřit jednotlivé neznámě veličiny ve větvích. Ty se počítaly právě z těchto tří smyčkových proudů, nebo řezových napětí a to dle toho, jaké smyčky, nebo řezy jsme v obvodu volili. Pokud se podíváme na metodu smyčkových proudů, vždy jednotlivé smyčkové proudy, které jsme si už vypočetli, odpovídali 3větvím v obvodu. A to podle toho, ve kterých větvích se neprotínaly dvě různé smyčky. V těch větvích, kde se dvě různé smyčky protnuly, bylo potřeba jednotlivé proudy těchto smyček od sebe odečíst, nebo je k sobě přičíst, dle toho, jestli směry těchto smyček byly ve stejném směru, nebo ve směru opačném. V poslední řadě nezbývalo nic jiného, než v případě metody smyčkových proudů vyjádřit neznáme napětí na ideálním proudovém zdroji a v případě metody řezových napětí vyjádřit neznámý proud na ideálním napěťovém zdroji.

Při řešení této práce bylo zřejmé, že všechny výpočty nemohou vyjít jako celé číslo. Proto bylo nutné, zobrazit výsledky ve tvarů pro nás vhodném. Pro tento případ jsem zvolil zápis výsledku

zaokrouhleného na dvě desetinná místa. K tomu jsem použil metodu *Math.round*, která je vyobrazena na obrázku 4.12. Tato metoda zaokrouhluje číslo umístěné v závorce za touto metodou a zaokrouhluje jej na celé číslo. Tudíž pokud bychom měli např. číslo 15,6368, metoda *Math.round* by toto číslo zaokrouhlila na číslo 16. Pokud bychom tedy toto číslo chtěli zaokrouhlit na dvě desetinná místa, bylo nutno toto číslo vynásobit číslem 100, jak je tomu na obrázku, tudíž by vzniklo číslo 1563,68. Pokud bych poté toto číslo pomocí *Math.round* zaokrouhlil, vzniklo by číslo 1564. Nyní toto číslo stačilo už pouze vydělit číslem 100, tím nám vzniklo 15,64 a tím je číslo 15,6368 zaokrouhleno na dvě desetinná místa.

```
Math.round(hodnota*100))/100);
```

Obrázek 4.12: Ukázka kódu pro zaokrouhlování čísel

4.5 Algoritmus sestavení rovnic

V této části projektu se zabývám vytvořením algoritmu, který na základě zadaného elektrického obvodu dokáže sestavit jednotlivé rovnice smyček nebo řezů dle I. a II. Kirchoffova zákona. Tak jako tomu bylo v kapitole 4.4, je i zde funkčnost sestavování založena na základních třech kombinacích jednotlivých smyček a 6 kombinací řezů, podle kterých se následně samotné rovnice sestavuji.

Algoritmus se skládá z celkem dvou částí. První část je jeden menší algoritmus, který sestaví základní rovnice dle směru smyček nebo řezů a pracuje tak, že projíždí větev od větve a dotazuje se, jaký prvek se na dané větvi nachází. Jakmile tento prvek najde, připíše jej do textového řetězce a tak to pokračuje, dokud není textový řetězec úplný. Pokud si vezmeme konkrétní příklad a budeme mít smyčku skládající se z tří větví, algoritmus se začne okamžitě zajímat o první větev a začne zjišťovat, jaký prvek se na ní nachází. Jakmile ho algoritmus najde, zapíše jej do textového řetězce a to samé udělá i s větví druhou. Správný prvek pak připíše na konec tohoto řetězce, hned za prvek druhý a takto to pokračuje i u větve třetí.

Takto algoritmus sestaví všechny rovnice, pro celý obvod a následně se začne provádět algoritmus složitější, u kterého je potřeba sestavené rovnice upravit tak, že vnitřní zdroje převedeme na pravou stranu rovnice a úbytky napětí nahradíme dle Ohmova zákona.

Tento algoritmus pracuje podobně jako algoritmus předchozí, ale zde bylo velmi důležité správně sčítat nebo odečítat jednotlivé hodnoty prvků. U předchozího algoritmu bylo vždy jedno, jestli hodnota u prvku bude kladná či záporná, jelikož tohle bylo pevně dáno vůči směru smyčky nebo řezu a počátečních směrů toků. U tohoto složitějšího algoritmu bylo nutné dotazovat se, jaké znaménko bylo v předchozích rovnicích u jednotlivých zdrojů, aby mohl toto znaménko obrátit při převodu na

druhou stranu rovnice. Nyní nastal problém, kdy program přesně nevěděl kolik prvků, se bude celkem nalézat za rovná se a tudíž, kdy bude potřeba nějaké dva prvky spojit pomocí znaménka plus. Proto bylo nutné se vždy tázat, jestli se za určitým prvkem ještě nějaký prvek nalézá a pokud by měl kladnou hodnotu znaménko plus zapsat, jinak by se mohlo stát, že na konci rovnice by vzniklo nevýznamné plus za kterým by se již nic nenacházelo. Proto oba tyto algoritmy dokáží sestavit jakékoliv rovnice celého obvodu, bez ohledu na to, co se v které větvi nachází.

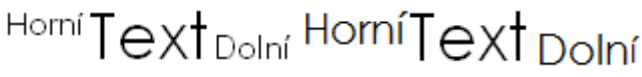
4.6 Algoritmus Tellegenovy věty

Ověření výsledků Tellegenovou větou, je nejsložitější algoritmus celého mého projektu. Zde bylo velmi důležité, z vypočtených výsledků správně určit, jakým směrem opravdu tečou jednotlivé proudy nebo napětí ve větvích a řezech, aby bylo možné určit, zdali výkony na jednotlivých prvcích budou kladné, či záporné.

Proto jsem sestrojil algoritmus, který dokáže z vypočtených hodnot zjistit právě tento směr toku tak, že zkoumá jednotlivé větve v obvodu a táže se, jestli jsou jednotlivé proudy nebo napětí kladné či záporné a na základě počátečních směrů toků poté směr obrátit, nebo jej nechat. Zde bylo také velmi důležité zohlednit situace, kdy by uživatel zadal zápornou hodnotu veličiny jako vstupní hodnotu a tím by tedy obrátil směr toku. Pokud tedy takto algoritmus určil správnost jednotlivých toků, následovalo v případě metody smyčkových proudů porovnání s napětím a v případě metody řezových napětí porovnání s proudy. Pokud by poté toky na jednotlivých prvcích tekly stejným směrem, výkon na daném prvku bude kladný, jelikož se prvek bude chovat jako zátěž, pokud by toky tekly směry opačnými, prvek se bude chovat jako zdroj a výkon bude záporný.

4.7 Písmo

Celý tento projekt se zabývá elektrickými obvodovými rovnicemi a k tomu patří i otázka dolních a horních indexů. Tohle byl dost velký problém, který se mi dlouho nedařil vyřešit. Nakonec jsem našel řešení, které spočívá v nahrazení dolních a horních indexů písmem, které právě tyto indexy připomíná. Jedná se o Superskriptové a Subskriptové znaky, neboli dolní a horní indexy, které nejsou součásti v běžné databázi fontů písmen. Proto bylo nutné tyto fonty vyhledat na internetu, které se zde dají běžně stáhnout. Tyto fonty jsou zde uloženy pod jmény *GG Superscript* a *GG Subscript*, které jsem následně nainstalovat do počítače. Po tomto kroku se fonty přidaly mezi ostatní fonty OS, a jaký je rozdíl mezi realizaci indexu pomoci těchto fontu a realizaci indexu pomoci funkce dolní nebo horní index, kterou známe např. z textového editoru Word, můžeme vidět na obrázku 4.13.



a) *b)*

Obrázek 4.13: Porovnání způsobů realizace dolních a horních index
a) Realizace za pomoci speciálních fontu, b) realizace za pomocí funkce

Používání této metody, bylo možno realizovat hned několika způsoby. Jedním takovým efektivním způsobem, bylo nastavení fontu na pevně daný text pomocí HTML příkazů. K tomu jsme museli použít třídu pro textové pole jménem htmlText, kterou můžeme vidět na obrázku 4.14. Tyto HTML příkazy se píši do hranatých závorek, kde v první uzavřené dvojici je definovaný příkaz jaký bude použit a v druhé uzavřené dvojici ukončení tohoto příkazu pomocí zpětného lomítka. Mezi těmito dvěma uzavřenými dvojici závorek se nachází text, v našem případě *1*, na který se provede tento zadaný příkaz. Ve výsledku se do textového pole vypíše *U* s dolním indexem *1*.

```
Text_Field.htmlText="U<font face='GG Subscript'>1</font>";}
```

Obrázek 4.14: Ukázka kódu pro nastavení dolního indexu na pevně daný text

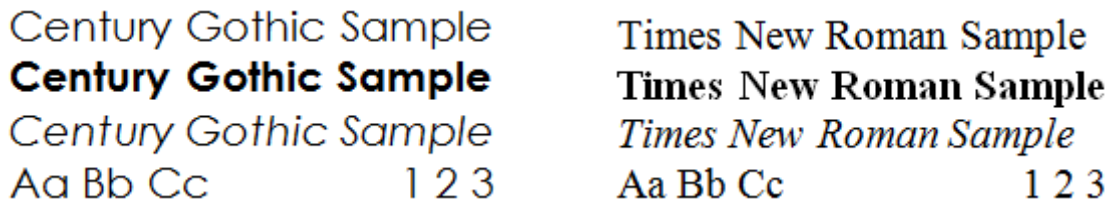
Druhý způsob se trochu lišil od toho prvního, ale používal taktéž HTML příkazů. Šlo o vyhledávání znaků v textu podle zadaných znaků, popř. řetězců znaků a následné nahrazení superscriptovým fontem. Tuto situaci můžeme vidět na obrázku 4.15, kde si v prvním řádku vytváříme proměnou *reSuper* typu *RegExp*, jedná se o regulérní výraz, kde mezi lomítka a závorky uvedeme jednotlivé znaky, nebo řetězce znaků, oddělenými svislými čarami, které budeme nahrazovat. Příznak *g* na konci řetězce značí vlastnost *global* a udává, odpovídání více než jen jedné shodě.

V druhém řádku se znova používá třída textového pole *htmlText*, kterému se přiřazuje textový řetězec *String1* datového typu *String*, ve kterém se třídou *replace* nahrazují veškeré výrazy v řetězci, které odpovídají regulérním výrazům v *reSuper*. Ty se následně nahrazují fontem GG Subscript. *$1* značí, že se jedná pouze o dílčí řetězec.

```
var reSuper:RegExp = /(1|2|3|4|5|6|7|8|9|R|G|X|1:|2:|3:)/g;
Text_Field.htmlText=String1.replace(reSuper, "<font face='GG Subscript'>$1</font>");
```

Obrázek 4.15: Ukázka kódu pro nastavení dolního indexu proměnlivého textu

Nyní, když už byly tyto indexy vyřešeny, vznikl další problém, a to takový, že u klasického stylu písma tyto indexy nešly moc vidět, a co se týče estetiky, také nebyly příliš vhodné. Proto bylo nutné najít takový styl písma, který by v tomto projektu vyhovoval nejlépe. V tomto projektu jsem použil font Century Gothic, který mi vyhovoval ze všeho nejlépe. Jedná se o bezpatkové písmo a rozdíl mezi standartním fontem Times New Roman, můžeme vidět na obrázku 4.16.



a) *b)*

Obrázek 4.16: Porovnání jednotlivých fontů písma

a) Font Century Gothic , b)Font Times New Roman

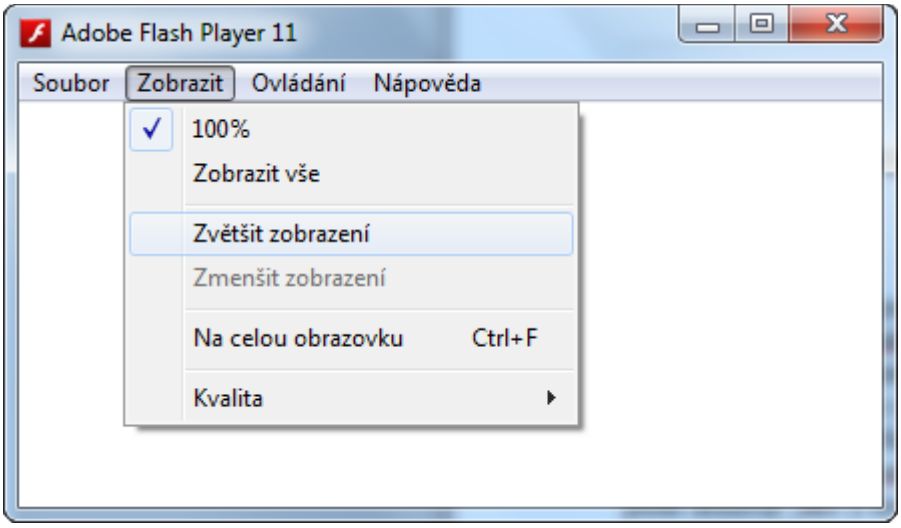
V posledním bodě bylo také velmi důležité udělat kurziva písma u jednotlivých veličin. Zde jsem využil opět HTML tágu, které bylo potřeba vložit přímo do textu. Pro tento tág oproti předchozím HTML tágům nebylo nutné přiřazovat třídu pro textové pole *htmlText*, ale stačilo přiřadit pouze klasickou třídu *text*. Následnou implementaci těchto tágu jsme provedli následujícím příkazem *<I>kurzivy text</I>,* kde na místo *kurzivy text* se napíše text, u kterého kurzivo, chceme použít.

5 Popis programu

5.1 Spuštění

Tento program pro výpočet MSP a MŘN je vytvořen ve formátu (*.swf) a je možné jej otevřít v programu Adobe Flash Player v nejnižší verzi 9, nebo v jiných programech, které jsou k dispozici ke stažení na internetu a právě tento formát podporují. Velikost obrazovky tohoto programu jsem volil tak, aby byla ideální pro displeje s uhlopříčkou 15,6“ a větší, kde pří 100% zobrazení nejsou jednotlivé obrázky nebo texty deformovány. Na menších displejích toto zobrazení bude deformováno, ale program Adobe Flash Player umožnuje uživateli zvětšit zobrazení v okně a následně po ploše srolovat jako u dotykových obrazovek. Vertikální ani horizontální posuvníky tento program nenabízí. Toto zvětšení zobrazení můžeme najít v záložce *Zobrazit*, která se nachází v horní části programu, což můžeme vidět na obrázku 5.1.

Aby se uživateli zobrazovaly dolní a horní indexy, je potřeba do počítače nainstalovat fonty GG Subscript a GG Superscript, jinak se tyto indexy zobrazovat nebudou. Viz kapitola 4.7.



Obrázek 5.1: Ukázka změny zobrazení v programu Adobe Flash Player

5.2 Zadání obvodu

Při spuštění programu se uživateli na displeji objeví základní kostra obvodu, kterou můžeme vidět na obrázku 4.3 a výzva pro uživatele, aby v každé větvi obvodu zadal právě jeden prvek. Jakmile uživatel označí jakoukoliv větev, tato výzva se na obrazovce ztratí. Tato výzva slouží k tomu, aby i uživatel, který se s tímto programem setkal prvně, věděl, co vůbec dělat.

Uživatel může docílit zadaného obvodu tak, že bude označovat jednu větev za druhou, čímž se mu vždy zobrazí napravo od této kostry pět obrázků jednotlivých prvků a podle toho, na který prvek uživatel klikne, tak ten se přidá do větve, kterou předtím vybral. Program také umožňuje vybrat více větví najednou a prvek se poté přidá do větví všech, které uživatel předtím vybral. Toto se nazývá editační režim.

Mimo tuto základní kostru se na displeji objevuje také tlačítko *Pokračovat*, které můžeme vidět na obrázku 5.2. Toto tlačítko slouží ke kontrole elektrické obvodu, zdali jej uživatel správně zadal. Podle požadavku této práce se musí v obvodu objevovat právě jeden ideální napěťový zdroj, právě jeden ideální proudový zdroj, právě jeden skutečný napěťový zdroj a právě jeden skutečný proudový zdroj. Proto bylo nutné kontrolovat, zdali byl tento požadavek splněn.

Obrázek 5.2: Ukázka tlačítka Pokračovat

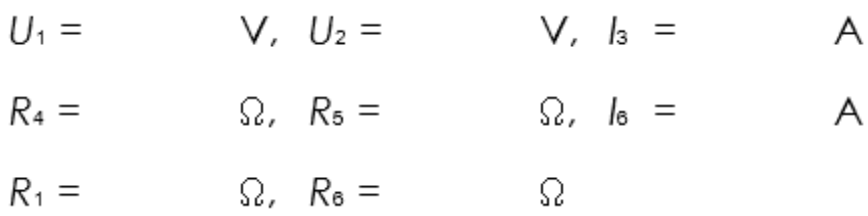
Tlačítko Pokračovat začne procházet celý obvod a začne zjišťovat, jestli byla tato podmínka splněna a zdali se v nějakých větvích právě tyto zdroje vyskytují. Pokud tomu tak není, vypíše se uživateli na obrazovce chybová hláška, která ho upozorňuje na to, aby tento požadavek dodržel, a tlačítko zůstává zatím neaktivní.

Taktéž se v obvodu kontroluje případ, kdyby uživatel nechal nějakou z větví prázdnou, tudíž nevybral by v ní žádný prvek. Pro tento případ se zobrazí nová chybová hláška, která uživatele o tomto problému informuje a tlačítko stále zůstává neaktivní, dokud uživatel chybu neopraví.

V posledním kroku se také kontroluje, zdali uživatel nevybral nějakou z větví a poté zapomněl vybrat nějaký z prvků, tudíž se stále nacházel v editačním režimu. V tomto případě je tlačítko stále neaktivní a neprovede nic, dokud program nevyhodnotí obvod, jako správně zadaný. Do té doby je toto tlačítko stále neaktivní.

5.3 Zadání hodnot

Pokud uživatel správně zadal elektrický obvod a následně stisknul tlačítko *Pokračovat*, zobrazily se na displeji textová pole pro jednotlivé prvky, které uživatel v obvodu vybral. Do těchto polí může následně vkládat hodnoty daných prvků, pokud však tyto hodnoty nevloží, automaticky se počítá s hodnotou 0. Takové zobrazení můžeme vidět na obrázku 5.3. Více o tom, jak správně zadávat hodnoty do těchto polí se dozvíte v kapitole 4.4.



Obrázek 5.3: Ukázka textových polí jednotlivých prvků v obvodu

Kromě těchto prvku se na displeji objevily i dvě tlačítka, tlačítko *Metoda smyčkových proudů* a tlačítko *Metoda řezových napětí*, které jsou vyobrazeny na obrázku 5.4. Tyto dvě tlačítka slouží k výpočtu obvodu dle metody smyčkových proudů nebo metody řezových napětí, podle toho, které tlačítko si uživatel vybere.

a) *b)*

Obrázek 5.4: Ukázka tlačítek Metody smyčkových proudů a metody řezových napětí
a) Tlačítko Metoda smyčkových proudů, b)Tlačítko Metody řezových napětí

5.4 Výpočet

Pokud uživatel zadal obvod, který chce počítat a zadal hodnoty jednotlivých prvků, nebrání nic v tom, abychom si jednotlivé výsledky mohli zobrazit. Ať už uživatel stiskne tlačítko *Metoda smyčkových proudů*, nebo tlačítko *Metoda řezových napětí* vždy se provedou tři nejdůležitější algoritmy celého programu a to algoritmus výpočtu, algoritmus sestavení rovnic a algoritmus Tellegenovy věty, na jejímž základě je celý program postaven. Tyto algoritmy jsou postupně popsány v kapitole 4.4, kapitole 4.5 a kapitole 4.6.

V prvé řadě se provede algoritmus pro sestavení rovnic a uživateli se na displeji vypíše rovnice pro jednotlivé smyčky, nebo řezy, dle toho co uživatel počítá. Tyto rovnice jsou sestaveny dle I. a II. Kirchoffova zákona. Pro metodu smyčkových proudů můžeme poté vidět toto zobrazení na obrázku 5.5 a pro metodu řezových napětí na obrázku 5.6. Obě tyto rovnice jsou sestavovány podle obrázku 4.4.

$$S_1:\ -U_1 + U_{R1} + U_2 + U_{X3} = 0$$
$$S_2:\ -U_2 + U_{R4} + U_{R5} = 0$$
$$S_3:\ -U_1 + U_{R1} + U_{R4} + U_{R6} + U_6 = 0$$

$$S_1:\quad R_1\cdot(I_{S1}+I_{S3}) = U_1 - U_2 - U_{X3}$$
$$S_2:\quad R_4\cdot(I_{S2}+I_{S3}) + R_5\cdot I_{S2} = U_2$$
$$S_3:\quad R_1\cdot(I_{S3}+I_{S1}) + R_4\cdot(I_{S3}+I_{S2}) + R_6\cdot I_{S3} = U_1 - U_6$$

Obrázek 5.5: Ukázka rovnice pro jednotlivé smyčky dle 2.Kirchoffova zákona

$$\check{R}_1:\ I_{X2} - I_3 + I_{G5} = 0$$

$$\check{R}_2:\ I_{G1} - I_1 - I_3 + I_{G4} + I_{G5} = 0$$

$$\check{R}_3:\ I_{G1} - I_1 - I_3 + I_{G6} - I_6 = 0$$

$$\check{R}_1:\ G_5 \cdot (U_{J1} + U_{J2}) = - I_{X2} + I_3$$

$$\check{R}_2:\ G_1 \cdot (U_{J2} + U_{J3}) + G_4 \cdot U_{J2} + G_5 \cdot (U_{J1} + U_{J2}) = I_1 + I_3$$

$$\check{R}_3:\ G_1 \cdot (U_{J2} + U_{J3}) + G_6 \cdot U_{J3} = I_1 + I_3 + I_6$$

Obrázek 5.6: Ukázka rovnice pro jednotlivé řezy dle 1.Kirchoffova zákona

Následně se právě z těchto rovnic sestaví matice, podle které se provádějí výpočty proudů v jednotlivých smyčkách, nebo napětí v jednotlivých řezech. Na obrázku 5.7 máme ukázku takovéto matice, která se pro obě metody liší pouze v číslech. Tyto čísla jsou pro naše záměry nepotřebné a slouží pouze jako ukázka.

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 6 & 4 \\ 1 & 4 & 11 \end{bmatrix} \cdot \begin{bmatrix} -4 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 7 \end{bmatrix}$$

Obrázek 5.7: Ukázka matice

Po vypočítání této sestavené matice se na displeji zobrazí jednotlivé proudy nebo napětí podle toho jakou metodu právě počítáme. Pro výpočet metody smyčkových proudů jsou tyto proudy vyobrazeny na obrázku 5.8 a pro výpočet metody řezových napětí na obrázku 5.9.

I_{S1}: -4 A I_{S2}: -0.44 A I_{S3}: 1.16 A

Obrázek 5.8: Ukázka vypočtených proudů pro jednotlivé smyčky v obvodu

U_{J1}: -2 V U_{J2}: 2.88 V U_{J3}: 0.96 V

Obrázek 5.9: Ukázka vypočtených napětí pro jednotlivé řezy v obvodu

Pokud máme tedy tyto jednotlivé proudy smyček a jednotlivé napětí řezu spočítány, není problém vyjádřit proudy nebo napětí v jednotlivých větvích a vypočíst neznáme napětí na ideálním proudovém zdroji v případě metody smyčkových proudů a neznámy proud na ideálním napěťovém zdroji v případě metody řezových napětí. Tyto proudy a napětí jsou následně vyobrazeny na obrázcích 5.10 a5.11.

I_1: -2.84 A I_2: 3.56 A I_3: -4 A
I_4: 0.72 A I_5: -0.44 A I_6: 1.16 A

U_X: 1.84 V

Obrázek 5.10: Ukázka vypočtených proudů pro jednotlivé větve obvodu

U_1: 3.84 V U_2: -2 V U_3: 1.84 V
U_4: 2.88 V U_5: 0.88 V U_6: 0.96 V

I_X: 3.56 A

Obrázek 5.11: Ukázka vypočtených napětí pro jednotlivé větve obvodu

V poslední fázi se provede výpočet Tellegenovy věty, který je vyjádřen jako součet všech výkonu na větvích aktivních a součet všech výkonů na větvích pasivních, který můžete vidět na obrázku 5.12. Součet těchto výkonů potom musí dávat hodnotu nula, čímž se ověří správnost výsledku. Podrobnější popis této věty můžete najít v kapitole 4.6.

Tellegenova věta:

$P_{aktivní}$: -17.36 W $P_{pasivní}$: 17.36 W

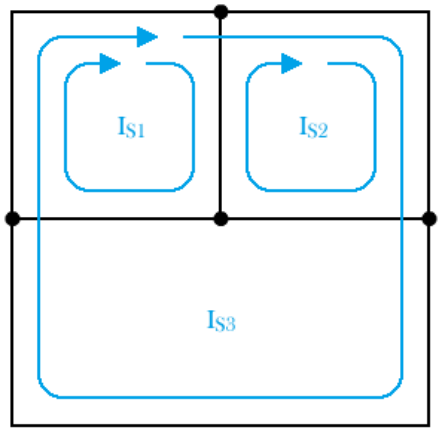
Obrázek 5.12: Ukázka Tellegenovy věty

Kromě těchto výsledků se na displeji objevilo tlačítko *Smyčky* nebo *Řezy*, dle toho jakou metodu počítáme a tyto tlačítka slouží pro zobrazení jednotlivých smyček, nebo řezů v obvodu, podle kterých byl obvod řešen. Tyto dvě tlačítka můžeme vidět na obrázku 5.13.

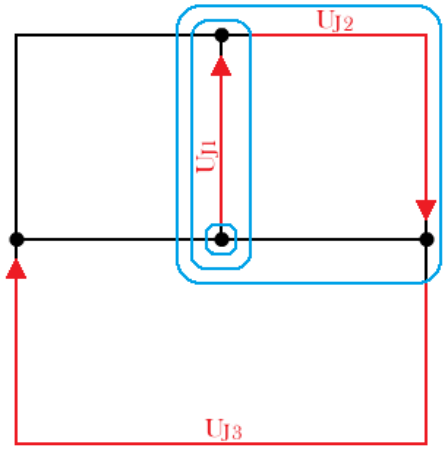
a) *b)*

Obrázek 5.13: Ukázka tlačítek Smyčky a Řezy
a) Tlačítko Smyčky, b)Tlačítko Řezy

Zobrazení, jak takové smyčky nebo řezy jsou v obvodu vyobrazeny, můžeme nalézt na obrázcích 5.14 a 5.15.



Obrázek 5.14: Ukázka zobrazení jednotlivých smyček v obvodu

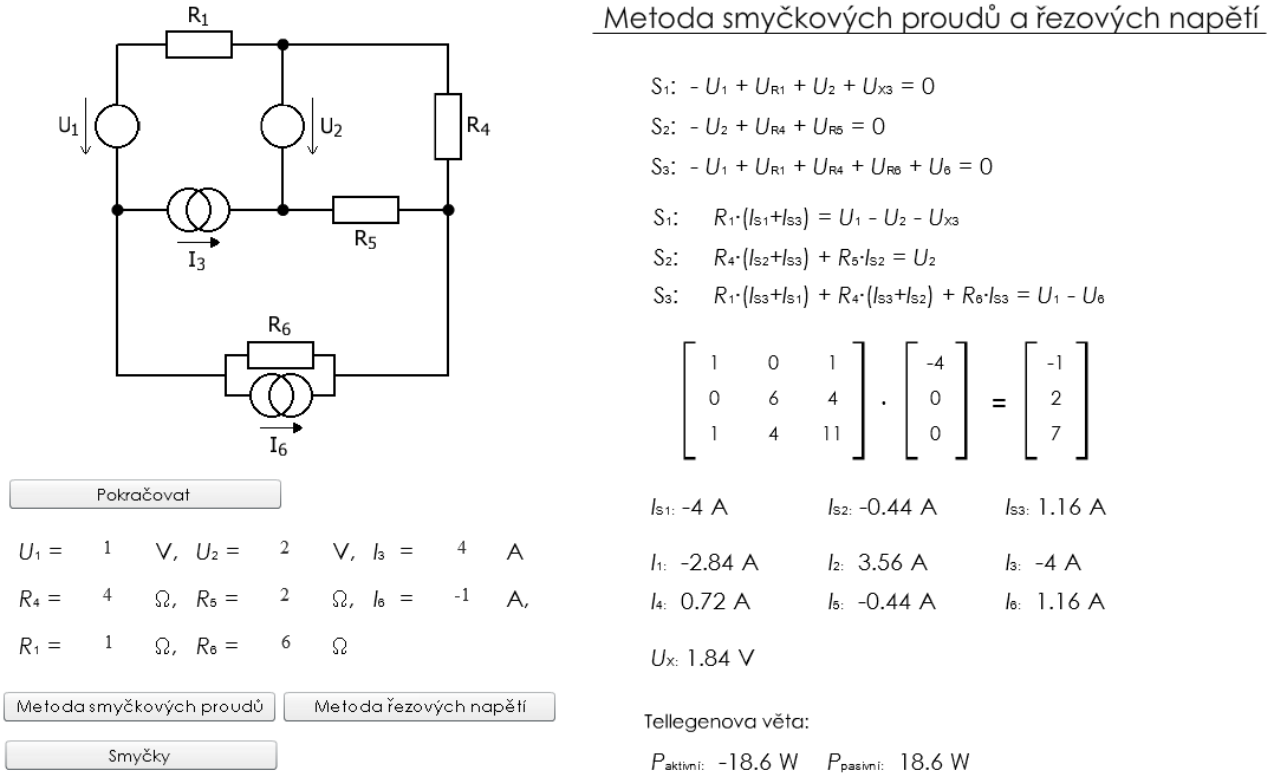


Obrázek 5.15: Ukázka zobrazení jednotlivých řezů v obvodu

V obou dvou případech, ať klikneme na tlačítko *Smyčky* nebo *Řezy* se vždy dané tlačítko přepíše na tlačítko *Zpět*. Toto tlačítko umožňuje přepnout se zpět do zobrazení s jednotlivými prvky v obvodu a to tak, že program si první musel zapamatovat, co se v něm nachází, aby následně mohl dané prvky zpět do obvodu zobrazit. Toto tlačítko můžeme vidět na obrázku 5.16.

Obrázek 5.16: Ukázka tlačítka Zpět

Pokud shrneme předchozí podkapitoly v kapitole 5, a spojíme je všechny dohromady, výsledek bude následně vypadat jako na obrázku 5.17, kde máme správně zadaný obvod, správně zadané hodnoty, sestavené rovnice dle I. a II. Kirchoffova zákona, následné sestavení matice a jednotlivé výpočty. V poslední řadě vypsání Tellegenovy věty. Pokud se podíváme na hodnotu proudu I_6 v textovém poli, můžeme vidět ukázku, jak obrátit směr toku právě tohoto proudu a to ze směru doprava, na směr doleva.



Obrázek 5.17: Ukázka celkového programu při výpočtu metody smyčkových proudů

5.5 **Kontrola výsledků**

Během kontroly výsledků se může stát, že výsledky proudu v jednotlivých smyčkách, nebo výsledky napětí v jednotlivých řezech budou rozdílné oproti výsledkům, které byly počítány například na papír a podle jiných smyček nebo řezů. Nicméně dopočítané proudy nebo napětí na jednotlivých spotřebičích už musí být správné, ale můžou se lišit ve znaménku. Proto je nutné podívat se na směry smyček nebo řezů a porovnat je s počátečními směry toků. Dle toho pak správně napsat znaménko.

6 Závěr

Naším úkolem bylo vytvořit výpočetní program, který by dokázal řešit lineární elektrické obvody podle metody smyčkových proudů a metody řezových napětí. Nejdůležitější součásti tohoto programu bylo umožnění uživatelům sami si volit obvod, se kterým by chtěli pracovat.

Po nastudování veškerých materiálů, co se týče teoretické části a tím i pochopení výpočtů metody smyčkových proudů a metody řezových napětí, bylo ještě potřeba seznámit se s vývojovým prostředím programu Adobe Flash Professional a s programovacím jazykem ActionScript, ve kterých jsem projekt realizoval. Po mém velkém úsilí se mi podařilo nejen celou látku pochopit, ale také dokončit i svůj výpočetní program. V závěru mé práce tento program nabízí uživatelům, sami si volit obvod, se kterým by chtěli pracovat, zadat hodnoty pro jednotlivé prvky a kontrolu jednotlivých rovnic se samotnými výsledky neznámých větvových veličin. Aby toho však nebylo málo, program sám provede kontrolu svých vypočtených výsledků a vypíše ji v podobě Tellegenovy věty. Krom tohoto, v poslední řadě program umožnuje nahlédnout do hlubší struktury řešení a umožňuje vykreslení jednotlivých smyček a řezů, podle kterých byl obvod řešen a umožňuje vypočíst 360 kombinací pro metodu smyčkových proudů a 360 kombinací pro metodu řezových napětí.

Dalším možným vývojem tohoto programu, může být umožnění uživatelům, aby si sami mohli volit samotnou kostru obvodu a následně i jednotlivé smyčky nebo řezy, podle kterých by byl obvod řešen. Pro naše potřeby je tento program však postačující a především studenti předmětu Teorie Obvodů, vyučovaný na naší fakultě, by mohli tento program velice ocenit při řešení různých projektů nebo domácích úkolů. Pozor, slouží pouze ke kontrole, jelikož v projektu není úplný postup řešení, ale pouze jeho výsledky, aby si studenti neušetřili veškerou snahu do tohoto předmětu

S volbou této práce jsem nadmíru spokojený a i přesto, že jsem nad touto práci trávil téměř většinu svého volného času, mě práce na tomto programu velice bavila. I přes časté problémy, ze kterých jsem někdy myslel, že už cesta nevede, mě nic neodradilo od toho, tento program dokončit. Většinu problému jsem měl především se samotným programovacím jazykem, jelikož spousta věci zde byla jiná, než jsem byl zvyklý u jiných programovacích jazyků. Pokud se však jednalo o problémy z oblasti teorie obvodů, obracel jsem se s problémy na mého vedoucího práce, který nikdy nezaváhal mi nepomoct. Tímto jsem mému vedoucímu práce velice vděčný.

Použitá literatura

[1] KIJONKA Jaromír a kolektiv. Teorie obvodů I. 1. VŠB – TUO, 2007. 196. ISBN ISBN 978-80-248-1488-9

[2] BLAHOVEC, Antonín. Elektrotechnika I. 5. Praha: INFORMATORIUM, 2005. ISBN 80-7333-043-1

[3] Steven E.Schwarz, William G.Oldman. Electrical Engineering. New York, 1984. ISBN 0-03-061758-8

[4] MIKULEC Milan, HAVLÍČEK Václav. Základy teorie elektrických obvodů. Praha: ČVUT, 2003. ISBN 80-01-01620-X

[5] JAREŠOVÁ, Miroslava. ELEKTRICKÉ OBVODY: Stejnosměrný proud. [online]. [cit. 2013-05-06]. Dostupné z: http://fyzikalniolympiada.cz/texty/elobvody.pdf

[6] GustavRobertKirchhoff. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-06]. Dostupné z: http://cs.wikipedia.org/wiki/Gustav_Robert_Kirchhoff

[7] KirchhoffovyZakony. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-06]. Dostupné z: http://cs.wikipedia.org/wiki/Kirchhoffovy_z%C3%A1kony

[8] AdobeFlash. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-06]. Dostupné z: http://cs.wikipedia.org/wiki/Adobe_Flash

[9] AdobeFlash. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-06]. Dostupné z: http://en.wikipedia.org/wiki/Adobe_Flash

[10] ActionScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-06]. Dostupné z: http://cs.wikipedia.org/wiki/ActionScript